Mitigating Disk Energy Management Delays by Exploiting Peer Memory

Guanying Wang, Ali R. Butt
Virginia Tech
Blacksburg, VA
Email: {wanggy,butta}@cs.vt.edu

Chris Gniady
University of Arizona
Tucson, AZ
Email: gniady@cs.arizona.edu

***Abstract*—Modern enterprises employ hundreds of workstations for daily business operations, which consume a lot of energy and thus have significant operating costs. To reduce such costs, dynamic energy management is often employed. However, dynamic energy management, especially that for disks, introduces delays when an accessed disk is in a low power state and needs to be brought into active state. In this paper, we propose System-wide Alternative Retrieval of Data (SARD) that exploits the large number of machines in an enterprise environment to transparently retrieve binaries from other nodes, thus avoiding access delays when the local disk is in a low power mode. SARD uses a software-based approach to reduce spin-up delays while eliminating the need for major operating system changes, custom buffering, or shared memory infrastructure.**

I. Introduction

Energy conservation in computer systems is driven by the positive financial and environmental implications, especially in servers [1], [2]. Large organizations often require shutting down workstations, unneeded servers and cooling systems overnight [3], as well as employ dynamic energy management to reduce energy costs. Dynamic management saves energy by identifying periods of inactivity for a device, and then keeping the device in a low-power state during such periods. Accurately predicting such idle periods [4] thus become a key research focus. However, these mechanisms still expose powering-on delays (e.g., disk spin-up delays), even if the predictions are correct and provide energy savings.

The challenge lies in keeping the system powered down for as long as possible, yet reducing the performance impact associated with delays on powering the device up when it is needed. Delays can significantly impact system performance, irritate users, and also reduce the energy savings since the system has to operate longer to satisfy user requests.

In this paper, we focus on reducing disk energy management delays because: disks are mechanical devices and thus expose large latencies when spinning up from low-power mode; they are significant energy consumers [5]; and disk energy management, e.g., shutting down idle disks [4], is a common practice present on almost every system in some form. To this end, we explore alternative ways of satisfying I/O requests destined for a typical desktop or workstation disk in low-power mode in enterprise environments. The goal is to reduce the spin-up delays by using existing resources present in such environments. Moreover, servicing I/Os from alternate sources provides opportunities for keeping the local disks in low-power mode, and may reduce energy consumption as a bonus.

To avoid spinning up the disk on arrival of user requests, and subsequently exposing spin-up delays to the users, we exploit the arrangement of local disks/file servers adopted in enterprise environments. Instead of always going to the local disk, or the centralized file server, we present System-wide Alternative Retrieval of Data (SARD) to retrieve application binaries from other workstations that are loosely arranged in a peer-to-peer (p2p) network. The key observation in SARD is that computers in enterprise environments are mostly uniformly configured to simplify system maintenance. As a result, the application binaries are identical across many peers, which allows sharing of the binaries among them.

SARD is designed in such a way that it: (1) does not require any custom buffering or shared memory infrastructure; (2) does not interfere with energy management of other systems; (3) does not require additional hardware resources; (4) requires few kernel modifications; and (5) allows participants to be loosely coupled and free to leave and join the system. SARD utilizes existing resources by transparently locating the workstation with requested binaries in the memory and transmits them to the machine that requested them. Furthermore, our p2p approach does not require fixed configurations and does not place any constraints on peers membership in the system. The individual machines can leave and join the system freely, significantly reducing system management that more tightly coupled systems, such as shared virtual memory, would require. The resulting design provides a low-overhead approach to minimizing energy consumption in enterprise environments.

II. Opportunities in Enterprise Environments

The following observations about large-scale enterprise environments serve as the enablers for SARD.

a) Energy management is prevalent: Large enterprises are actively pursuing energy management of employees' workstations for monetary savings. A popular dynamic energy management technique is to shut the disk down after a period of idleness. However, spinning-up disks to service later I/O requests can typically expose multi-second delays to the users.

b) Similarly maintained systems: Large computing infrastructures, especially academic setups, keep the systems

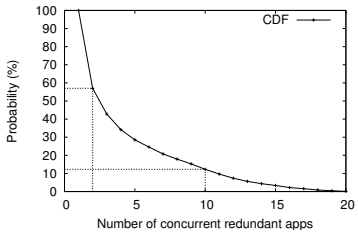

Fig. 1. Concurrent redundant apps.

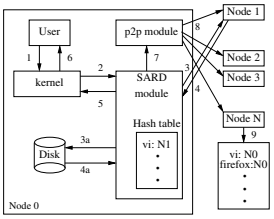

Fig. 2. SARD Architecture.

mostly uniform to simplify management: they run the same OS and set of applications, usually on similar hardware.

c) User data on central file servers: Persistence is crucial for user data, therefore it is periodically backed up. To simplify the backup process and to provide users transparent access to data from any workstation, many setups provide central storage for user data, which is more reliable and cost effective than backing up individual systems. The local disks are typically utilized to boot OS and supporting temporary scratch space.

d) Similar system usage: To investigate how often different machines in an enterprise use the same binary, we conducted a study using 20 of our departmental machines used by students for typical desktop use. For a period of 13 days, we recorded the applications that are running on each machine every 5 seconds. Next, we determined how often different machines run the same applications.

Figure 1 shows the cumulative distribution of the number of times when different machines are running the same applications. For the studied environment, 57% of the time two or more machines were running an application concurrently, and 12% of the time more than 10 copies of an application were running concurrently on different machines. Assuming a similar application usage distribution, and extrapolating these results[1] show that in a medium-scale setup with 107 machines more than 99% of the time an application will be running on at least two machines and thus can be serviced remotely. Consequently, such environments can benefit from SARD.

III. Design

In SARD, all machines join a p2p overlay network, which enables them to interact with each other in a decentralized fashion using the DHT [6] paradigm. Participants run our software that advertises their in-memory applications to others via the overlay. Advertisements enable participants to learn what applications (or parts thereof) are available in memory of peers. When an application is executed on a node, it can use the remote availability information and decide whether to retrieve the application from the local disk or remote memory. Servicing requests from remote memory helps avoid spin-up delays of powered down disks, and can improve energy savings by keeping disks in low-power mode longer.

[1]Analysis shows that for a setup of n ($n >> 19$) nodes, the probability that two copies of an application are running at the same time is $1 - \prod[(1 + (n-1)p_i)(1-p_i)^{n-1}]$, where p_i is the probability that an application i ($0 < i \leq n$) is running on a node at a given time. A detailed derivation is out of scope of this paper.

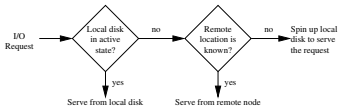

Fig. 3. Local vs. remote retrieval heuristics in SARD.

Figure 2 shows the architecture of SARD. The only kernel modification is to intercept and reroute disk I/O requests to the SARD module. After intercepting an I/O call (in the `read_pages()` function of standard Linux kernel) (Step 1 and 2), SARD checks the hash table to determine alternative sources for serving it. If a remote source is found, a UDP message requesting the image is sent to that node (Step 3). The corresponding SARD UDP server on the remote node receives and serves the request. Once a reply containing the requested image is received back at the requester (Step 4), the image is returned to the kernel (Step 5) just as if the request was serviced from the local disk (e.g., Step 3a and 4a). The request is finally returned to the user (Step 6). A loaded application is then advertised to other nodes (7, 8, 9).

Several factors affect the decision of whether to retrieve an application from the local disk or from remote memory. We use the intuitive set of heuristics shown in Figure 3, to drive SARD. The goal is to use the local disk as much as possible, but to avoid spinning it up if it has already been spun-down.

IV. Evaluation

We use Dell PCs, with an Intel 2.4 GHz dual core processor, 4 GB RAM, and a high-end Seagate 250 GB hard disk, connected using 1 Gbps Ethernet for our evaluation.

A. Implementation Results

SARD is implemented using about 2300 lines of C code. Additional 1200 lines of Java code are used to implement the p2p advertising daemon using FreePastry [6].

1) Remote Binary Serving: Modern operating systems load portions of applications from disk on-demand. We model this behavior in the controlled experimental setting by using PostMark to perform random I/O, in essence emulating on-demand random page loads of varying lengths. For each case, we measured the time it would take to service the request locally from disk or from remote memory. Figure 4 shows the ratio of the time used for servicing a request locally compared to that served remotely. Note that for these measurements the disks were spinning and in ready state. We observe that for smaller files, the disk performance is poor compared to remote retrieval – servicing from disk takes order of magnitude longer compared to over the network. The comparative benefit from remote retrieval is somewhat reduced for larger file sizes because the time to retrieve data from the disk improves significantly with increasing file sizes, i.e. for large sequential accesses.

2) Impact of SARD on Remote Machines: Next, we study the impact of SARD on remote node performance. First, we determined how a node's overall performance is impacted when servicing varying rates of page requests. For this purpose, we designed a benchmark that generates a controlled

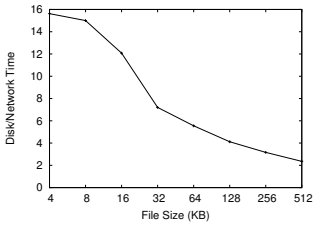

Fig. 4. The ratio of local access time compared to serving the binaries remotely.

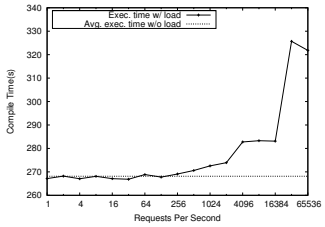

Fig. 5. Impact of servicing remote memory requests.

TABLE I
REQUESTS-PER-SECOND FOR VARIOUS APPLICATIONS, AND THEIR IMPACT ON REMOTE NODE PERFORMANCE.

Application Name	Requests per Second	Impact on Remote Node
cscope	112	0%
`make`	6.25	0%
PostMark (8 KB)	530	0.90%
PostMark (32 KB)	1073	1.63%
PostMark (128 KB)	2064	2.14%
PostMark (512 KB)	2541	2.94%

number of remote page requests at one of the test machines. On the other test machine, we compiled the Linux kernel and observed the compilation time for each case as we increased the number of requests generated per second from 1 to the extreme case of 65536. Figure 5 shows the results. The horizontal line shows the average time it takes to compile the kernel on a standard setup without any remote load. We observe that up to 256 requests per second are serviced without any observable performance degradation, and only 5.59% degradation is observed when as much as 16384 requests are serviced per second.

Second, using the above information, we determined how several test applications will affect remote nodes. For this experiment, we use: a *cscope* [7] query on Linux 2.6.22.9 source code, *make* to compile the same kernel, and PostMark with different file sizes. We observed the average rate of remote page requests issued by these applications. Table I shows the request rates. We then used the load impact numbers of Figure 5 to estimate the impact of the studied applications on a remote node serving the requests. In particular, observe that both *cscope* and `make` incur negligible overhead, and PostMark (8KB) that models on-demand application loading incurs less than 1% overhead.

B. Simulation Results for SARD's Energy Impact

Next, we present a simulation and implementation study that shows SARD's potential for energy savings.

1) Methodology: Detailed traces of user-interactive sessions for each application were obtained by a `strace`-based tracing tool [8] over a number of days. We used a Western Digital Caviar WD2500JD in our simulation with a spin-up time of about 9 seconds from a sleep state [9]. Table II shows six desktop applications that are popular in the enterprise environments and used in this study. The table also shows trace length and the details of I/O activity. Read and write

TABLE II
THE NUMBER AND DURATION OF TRACES COLLECTED FOR THE STUDIED APPLICATIONS.

Appl.	Trace Length [hr]	Number of		Referenced [MB]	
		Reads	Writes	Reads	Writes
mozilla	45.97	13005	2483	66.4	19.4
mplayer	3.03	7980	0	32.3	0
impress	66.76	13907	1453	92.5	40.1
writer	54.19	7019	137	43.8	1.2
calc	53.93	5907	93	36.2	0.4
xemacs	92.04	23404	1062	162.8	9.4

requests satisfied in the buffer cache are not counted, since they do not cause disk activity.

2) Energy Consumption: Serving the I/O from remote machines increases the length of idle periods by eliminating spin-ups required to serve the I/O requests from the local disk. Table III illustrates the impact of serving I/O requests on the length of idle periods. It shows the number and average length of idle periods for varying fractions of requests served by the remote machines. The case of 100% of requests served locally illustrates the standalone workstation that serves all requests from the local disk. By serving more and more requests from other workstations the number of idle times is reduced since the idle periods are concatenated resulting in fewer and longer periods. In the case of 1% of requests served locally, the average number of idle periods is reduced by 73.2% and the average length is extended by 205.5%. Based on our study of Section II, 1% of requests served locally is a reasonable number for a medium-scale setup (e.g. with more than 107 workstations), since many will have standard applications loaded in memory. In addition, we show results for serving 2%, 5%, 10%, and 15% which may be encountered for a small number of workstations in the network.

Reduction in number of periods and lengthening the duration of the idle periods has twofold impact on energy efficiency. First, fewer number of periods indicates that there are fewer spin-ups required to serve the I/O requests resulting in lower energy spent on powering up the devices and shutting them down. Second, longer idle periods will allow the disk to remain in a power saving state also reducing energy consumption. These can be seen in Figure 6, which shows distribution of the local disk energy consumption among three categories: Busy – due to serving the I/O requests, Idle – due to waiting for more requests to arrive during timeout interval, and Power-Cycle – due to shutting down and spinning up the disk. We show numbers normalized to the case when a standard energy saving mechanism is used in a stand-alone system, i.e., with 100% requests served locally. All of the states are impacted by SARD. We first observe that energy spent serving I/O requests is not significant since most of the applications are interactive with long user think times or they are accessing user files that are mounted on a remote file server. The two largest components are power-cycle and idle energy. The average fraction of energy spent on spinning up and shutting down the disks in the case of local disk only is 69.3%. The energy is reduced as we serve more and more from remote machines and reaches the average fraction of

TABLE III

NUMBER AND AVERAGE LENGTH OF APPLICATION IDLE PERIODS, AND DELAY DUE TO DISK SPIN-UP, AS INCREASING NUMBER OF REQUESTS ARE SERVICED REMOTELY. NOTE THAT FOR *Mplayer* UNDER ALL CASES THE NUMBER AND DURATION OF IDLE PERIODS IS 4 AND 2713 SECONDS, RESPECTIVELY, AND THE DELAY IS 36 SECONDS.

% of Reqs. Served Locally	Mozilla			Calc			Impress			Writer			Xemacs		
	Idle Prds.	Length [s]	Total Delay [s]	Idle Prds.	Length [s]	Total Delay [s]	Idle Prds.	Length [s]	Total Delay [s]	Idle Prds.	Length [s]	Total Delay [s]	Idle Prds.	Length [s]	Total Delay [s]
100	165	985	1485	150	1283	1350	227	1048	2043	136	1423	1224	95	3477	855
15	102	1601	918	89	2170	801	122	1959	1098	88	2206	792	59	5604	531
10	88	1858	792	77	2511	693	110	2174	990	80	2427	720	56	5906	504
5	82	1995	738	70	2763	630	87	2752	783	70	2776	630	49	6751	441
2	58	2825	522	52	3724	468	66	3631	594	51	3814	459	41	8070	369
1	49	3346	441	34	5701	306	45	5330	405	40	4867	360	38	8708	342

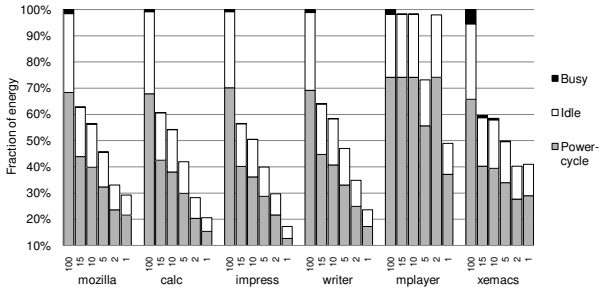

Fig. 6. Breakdown of energy savings as more and more requests are serviced from remote nodes normalized to the case of a standalone system (100% local requests).

22.2% for the case of only 1% of requests served locally. Similarly, the time spent in idle is reduced due to fewer timeout periods encountered as we increase fraction of requests served remotely. The average fraction of energy consumed at idle is 28.8% for all requests served locally and is reduced down to 7.9% for when serving only 1% of requests locally. Moreover, compared to an always-on scheme with no energy saving mechanism, the 100% approach provides an average savings of 80.6%, which is further improved to an average of 81.7% for the case with 1% local requests.

Fewer needed spin-ups result in shorter overall delays exposed to the user. Next, we measure this effect. Based on our observations of the network traffic of our test machines, we assumed network latencies of a 1 Gbps Ethernet connection with at most 20% degradation due to contention modeled randomly. Table III illustrates the total delay exposed to the user as we vary the fraction of I/O requests served locally. The average delay across applications is reduced from 1165.5 seconds for all requests served locally to 312 seconds, i.e., a 73% reduction, when we only serve 1% of requests locally. Reduction in delay has two benefits. First, the user experience is improved since the user will see fewer lags due to disk spinning up. As a result, the user is more likely to use energy management techniques as opposed to turning the energy management off to prevent the irritating delays. Second, the shorter delays will allow the user to accomplish the task quicker, which increases the efficiency of the system.

We also studied the commonly used energy-delay product (EDP) for our setup. We found that SARD reduces energy-delay product by 81.63% on average, compared to the always on case.

C. SARD Case Study

We studied the energy saving and performance impact of SARD using 10 of our departmental machines used for typical desktop use. Each machine has a an Intel Pentium 4 3.0 GHz processor, 1 GB RAM, 40 GB hard disk, and is connected via 1 Gbps Ethernet. For a period of two week, we traced the system usage of the workstations. Subsequently, we replayed the traces on the systems, measuring the energy consumed by the machines (using Watts up? PRO power meters). Next, we configured the machines to run SARD, and replayed the traces and once again measured the energy. The total energy consumed by the machines over the duration of the week reduced from 165312 Watt-hour (Wh) to 156895 Wh, a saving of 5.1%.

V. CONCLUSION

In this paper, we have presented the design and evaluation of SARD, a p2p-based system that mitigates delays associated with disk energy management by allowing sharing of in-memory application images across peers. Our evaluation of SARD using both a real implementation study and simulations demonstrates that SARD can serve as a practical and effective tool for mitigating energy management delays, which also improves energy efficiency in enterprise environments.

REFERENCES

[1] J. Chase, D. Anderson, P. Thackar, A. Vahdat, and R. Boyle. Managing energy and server resources in hosting centers. In *Proc. SOSP*, 2001.

[2] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Proc. COLP*, 2001.

[3] Tufts. Computers and energy efficiency. Online Specification, 2008. http://www.tufts.edu/tie/tci/Computers.html.

[4] Chris Gniady, Ali R. Butt, Y. Charlie Hu, and Yung-Hsiang Lu. Program counter-based prediction techniques for dynamic power management. *IEEE Transactions on Computers*, 55(6):641–658, 2006.

[5] Qingbo Zhu, Zhifeng Chen, Lin Tan, Yuanyuan Zhou, Kimberly Keeton, and John Wilkes. Hibernator: helping disk arrays sleep through the winter. In *Proc. SOSP*, 2005.

[6] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM Middleware*, 2001.

[7] Joe Steffen and Hans-Bernhard Bröker. CSCOPE, Jan 2008. http://cscope.sourceforge.net/.

[8] Ali R. Butt, Chris Gniady, and Y. Charlie Hu. The performance impact of kernel prefetching on buffer cache replacement algorithms. *IEEE ToC*, 56(7):889–908, 2007.

[9] Igor Crk and Chris Gniady. Context-aware mechanisms for reducing interactive delays of energy management in disks. In *Proc. USENIX ATC*, 2008.